CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Biblioteca Digital do IPB

Optimization of the IST Production domain through industrialization

Elisete Reis

Dissertação apresentada à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação.

Trabalho orientado por:

Rui Pedro Lopes

Bragança

2016

Optimization of the IST Production domain through industrialization

Elisete Reis

Dissertação apresentada à Escola Superior de Tecnologia e Gestão de Bragança para obtenção do Grau de Mestre em Sistemas de Informação.

Trabalho orientado por:

Rui Pedro Lopes

Bragança

2016

Abstract

Nowadays, technology trends are forcing enterprises to rethink their business and applications, increasing their complexity and dependencies. Enterprises require Information System Technologies (IST) departments to manage the information systems that support their core business. The complexity and number of the solutions used have increased and are in constant evolution. Therefore, Information and Communication Technology (ICT) professionals face challenges everyday to keep up with such evolution and related support tasks. To cope with these requirements, IST departments created specialized teams for different layers of the implemented solutions. The inter-dependency of those layers and teams slows down the response time, because the system and applications administration activities require the interaction of diverse teams and professionals, from simple tasks, to more complex ones (not only the solutions complexity augmented, but also processes need to be synchronized between different teams). In such scenario is difficult to keep an updated documentation of systems state, to control workflow and to have traceability of all changes.

One of the teams involved in these dynamics is the Production team, that is responsible of supporting applications during all its life cycle phases, and thus needs their processes and way of working to be agile. This dissertation analyses the scenario of the Production domain in large enterprises and how the principles of industrialization can be used to improve performance. It also makes an analysis of tools and methodologies that could instantiate such principles. Other inherent objective is to research a solution for the Production team be able to ameliorate automation levels (maintaining layer separation and the responsibilities of each team), enable traceability, and reduce response time.

Thus, framework and some processes are defined, a testing scenario is created and normal activities of the Production team are simulated.

Resumo

Hoje em dia a evolução tecnológica levam as empresas a repensar o seu negócio, aumentando a sua complexidade e dependências. As empresas dependem de departamentos de Tecnologias dos Sistemas de Informação (TSI) para gerir o sistema de informação que suporta o seu negócio. As soluções informáticas utilizadas estão em constante evolução e o seu número e complexidade tem aumentado. Assim, os profissionais têm de responder a um conjunto alargado de desafios para manter todas estas soluções atualizadas e garantir o seu suporte. Os departamentos de TSI cresceram para responder aos requisitos e foram-se criando equipas especializadas, responsáveis por diferentes camadas das soluções implementadas. A interdependência das camadas e das equipas aumenta o tempo de respostas, porque as tarefas de administração de sistemas e aplicações, simples ou complexas, requerem a interação de diversos profissionais e equipas. Neste tipo de cenário é difícil manter documentação atualizada, controlar o fluxo de trabalho e ter rastreabilidade de todas as alterações. A equipa de Produção, que é responsável por manter as aplicações durante todo o seu ciclo de vida, precisa que a sua forma de trabalhar seja ágil. Esta dissertação analisa o cenário, do domínio da Produção, em grandes empresas e verifica de que forma é que os princípios da industrialização podem ser utilizados para melhorar o desempenho. Investiga também ferramentas e metodologias que poderão instanciar os princípios identificados. Este trabalho procura uma solução, para a equipa de Produção, que permita melhorar o nível de automação, mantendo a separação de camadas e responsabilidades de cada equipa, permitindo a rastreabilidade e redução do tempo de resposta. Definiu-se uma estrutura e processos que suportam esta solução, criou-se um cenário de testes e simularam-se algumas atividades da equipa de Produção.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Prof. Rui Pedro Lopes for his motivation, patience, time and energy. His guidance, ideas and feedback have been absolutely invaluable. I would also like to thank all my teachers for their time and availability, my friends and my colleagues for their support and encouragement. And finally, I would like to thank my amazing family and boyfriend, for their unconditional trust, love and unfailing emotional support. I undoubtedly could not have done this without you.

Contents

List of Figures

List of Listings

Glossary

API Application Programming Interface. 28

CMFs Configuration Management Frameworks. 25, 31, 60

DNS Domain Name System. 2, 13

DRP Disaster Recovery Plan. 16, 21

ED Exploitation Documentation. 13, 15

HTML Hypertext Markup Language. 38

HTTPS HyperText Transfer Protocol Secure. 13

ICT Information and Communication Technology. v

IP Internet Protocol. 2

IS Information System. 1, 2

IST Information System Technologies. v, 1–4, 9, 13, 59

IT Information Technology. 4, 20, 22, 24, 25, 27, 60

OAT Operational Acceptance Testing. 9, 14

PDF Portable Document Format. 38

SLA Service-Level Agreement. 10, 21

SSH Secure Shell. 24, 25, 31

SSL Secure Sockets Layer. 14

SSO Single Sign-On. 2

TAD Technical Architecture Documentation. 11–13, 15, 21

TID Technical Installation Documentation. 13, 14, 21

TSI Tecnologias dos Sistemas de Informação. vii

UAT User Acceptance Testing. 10, 14

VM Virtual Machine. 13, 22

YAML Ain't Markup Language. 36

Chapter 1

Introduction

Through the years, technology has become more accessible, powerful, and more user-friendly. It is now a part of our daily life, and we constantly rely on it to manage information and to support many daily activities.

Enterprises started using IST to automate and dematerialize operations and processes. Nowadays, they use it in every process, demand, or service, and most tasks depend on informatics applications, supporting their business. Furthermore, recent social networking trends are forcing enterprises to rethink their business and applications, increasing their complexity and dependencies. Keeping those systems up-to-date, with continuous improvements, is a big challenge, which caused the appearance of applications and technological solutions to enable a more integrated, productive and manageable maintenance of all those applications.

Given that requirements are changing, enterprises are creating and developing their own IST departments. An IST department is composed by different teams and offices, with internal and/or outsourced personnel, typically responsible for some layers or components of the Information Systems (ISs).

Maintaining those professionals and teams working together is a huge challenge. In fact, their purposes and goals may be different, and incompatibilities may emerge. For example, the main purpose of one office might be the stability and security of the system, whereas another is intended to make the necessary changes as fast and cheap as

possible. Such conflicts can cause delays in implementing changes, eventually leading teams to blame each other. That explains the need to set processes and methods so as to permit communication and collaboration between organization collaborators, as well as supporting, updating, deploying, and keeping all the components working at their best level.

IST departments are organized into different domains which are adjusted into one or more offices depending on the scale of enterprises needs. To support the IS, the following domains are usually present:

- Project Management Office: Manage projects and coordinate processes, plans the different phases with each office, provides updated documentation, manages communication between all the involved parties and has the main objective to comply with the planned budget.

- Security / Network: In charge of the internal network infrastructure, internet services and security related operations. Responsible for supporting the core network services, such as Domain Name System (DNS), Single Sign-On (SSO), certificate and Internet Protocol (IP) address management, collaborating to improve services configuration when required. Enforce security mechanisms to protect external access to applications.

- Office Suite: Deals with purchases and provides all programs (software, tools, modules, or others) necessary to all enterprise users, also ensuring updates, upgrades, and compatibility between them. Usually in charge of formation on internal productivity softwares.

- Helpdesk: Main contact of the Information Systems Department with the users, handling and managing incidents, detecting and giving the first level support or redirecting to second level when is out of their competences.

- Production: Responsible for the applications life cycle, from the specification phase, through the environments creation and maintenance, until they are decommissioned.

Depending on the scenario, the outcome of an application's production can lead to produce and support the following environments: *development, qualification, integration, staging, pre-production and production*. The base line of each environment is provided by Operations.

- Operations: Also called Exploitation, is responsible for providing and maintaining a secure and stable baseline of diverse environments and their supporting infrastructure. The environments depend on the scenario. Usually Operations deals with the operating system administration of all environments, and monitoring the production. In addition, it is also responsible for enforcing the exploitation's procedures like backups, automation and supervision, managing and controlling all assets in the data center.

The work described in this dissertation focus on the Production domain, within the IST, and the way it interacts with the other domains, mostly Operations which, in some enterprises are integrated in the same office. The difficulties that arise from their responsibilities lead to the creation of solutions that enable teams to be more productive and to wisely exploit their resources. Some examples include using virtualization to consolidate their systems and more easily manage these systems, or using cloud based solutions to provide elasticity or allocating resources as needed.

These solutions enable reducing the response time of some tasks. However, they do not solve problems like making changes, updates, synchronization between test and production systems, execution of commands or execution of bulk tasks. They also do not support service orchestration or tasks execution that have dependencies on services from different systems.

A large enterprise with several distinct teams faces several challenges and often have different requirements for solutions in order to enable task automation and improving systems state, services, and component documentation.

This work focus on the Production domain, with its particular challenges and the test of the adoption of an industrialization philosophy, so that to evaluated its contribution

to minimize response time. We research how to implement a framework based on this philosophy, as well as the related technologies and tools that can support the processes and methods of the framework.

1.1 Goals

The fundamental goal of this work is to identify the weaknesses and constrains of the Production domain within the IST department, in order to propose a framework that can minimize them. Therefore the following objectives are outlined:

- Characterize the Production domain;
- Understand the constrains and weaknesses related to their responsibilities;
- Study how industrialization concepts can be applied to the Production domain;
- Analyze technologies and tools that can support our study;
- Propose a Framework;
- Create a test environment to analyze the use of the instantiated framework in regular Production activities.

1.2 Document Structure

This document is structured in five chapters, starting with this introduction. Chapter two provides a background analysis concerning Production, how it interacts with the other IST domains and identifies the challenges of working on a such complex structure. It also describes the industrialization philosophy and how it can be used in Information Technology (IT) Production. Technologies and tools are also studied to understand how the framework can be instantiated.

Chapter three presents the proposal. The approach used tries to reduce bottlenecks, related with the interdependence with other domains, the lack of normalization and standardized processes.

On chapter four, a testing environment is presented to explore and some common Production operations in order to evaluate the viability of the framework.

Chapter five gives a discussion about the proposed framework and analyzes some of the decisions made in the instantiation. It presents the general results and contributions as ideas for future work.

Chapter 2

Background

The adoption of information technologies on various professional areas in industry and in services made data storage and processing essential for the provisioning of goods and services. Data management is performed with specialized applications, with specific life-cycle that include the specification of those applications, the creation and maintenance of their execution environments, as well as their decommission.

The development or acquisition of applications are initiated by the Product Owner. This is a team of people with functional responsibilities, meaning that they define the functionality of applications. They define the contract terms for the new application or for the update of existing applications, the budget, and the deadline for it to be available to the end users. They intervene in the functional tests and pass the development or acquisition process to the Project Owner.

The Project Owner, according to the application's requirements, appoints a Project Manager to supervise all the process. The development may be internal or it may be outsourced, guided by the contract terms and budget. Either way, the application development is controlled by the Project Manager, ensuring that the schedule and the requirements are met.

The Project Manager, with the assistance of the Production Architect, create the technical documentation, defining all the security, compatibility, and dependency requirements. He also plans all the steps and schedule until the application is put in production.

Both the functional contract terms and the technical documentation, a complete acquisition or development contract is made. This will be used to outsource the development or to guide the internal development (Figure 2.1).

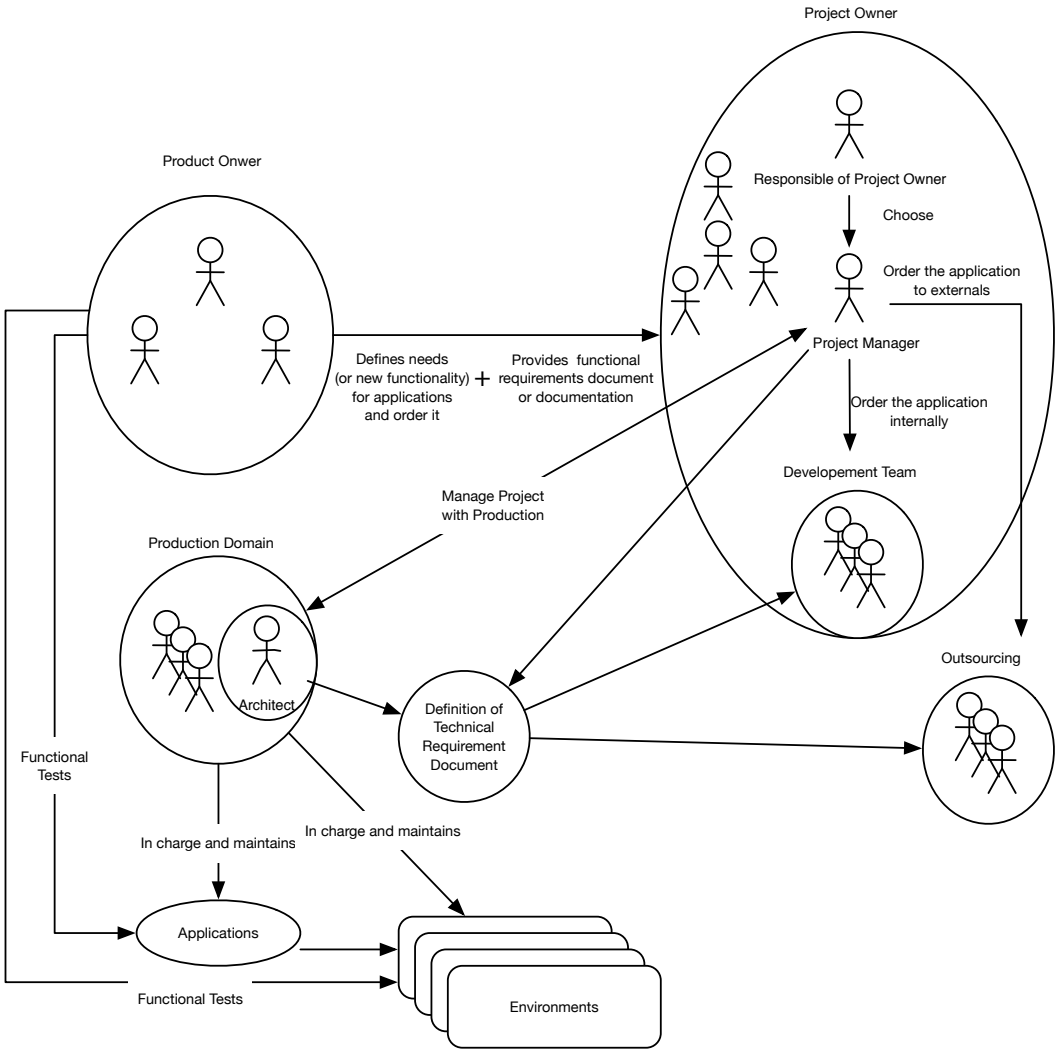

Figure 2.1: Interaction between actors

Most of the entities referred in this document are teams of people, such as the Product Owner, Project Owner, Production Domain or Development team. The Production Architect and the Project Manager are individuals, that manage the milestones, schedule, budget and contract terms.

2.1 Production Domain Activities

The Production domain of the IST department is responsible for managing all the steps of the applications' life-cycle. It is responsible for keeping applications constantly available, and to constantly the perennially, its consistency and availability of its data.

2.1.1 Environments

The work performed in the Production domain is closely associated to the applications' life-cycle. During the life-cycle phases, several environments are necessary. Their number depend on the size of the enterprise and on the application type. With a few exceptions, most of them require the following environments:

- Development: used to develop the application or to add new functionalities to an existing application. It also requires changing and updating the test units to match the required functionality, so that the contract terms are verified.

- Integration: this environment is used to put together all the different modules developed previously, and to confirm that they compile and that the integration tests are successful. This environment is more commonly used in bigger projects, usually requiring different teams to develop the modules. For smaller projects, the integration and the associated tests are made in the development environment.

- Staging: after successful integration, the application is installed according to the provided documentation. This environment is used to make the acceptance tests. First, the Production domain team performs the technical staging, with the Operational Acceptance Testing (OAT), which is a common type of non-functional software testing that covers key quality attributes of functional stability, portability and reliability (testing the installation, compatibility, recovery, regression, security, performance, supervision, and others). Each test must respect the technical architecture demands. After the OAT, the Product Owner conducts the functional

staging using the User Acceptance Testing (UAT), which consists of a process to verify that a solution works for the user and that it fulfills all the functional requirement of the contract terms.

- Qualification: this environment is created for critical applications, composed of several modules and used by different types of users. These need to be tested by different groups (for each module) and stay on production schedule. Qualification may also need several identical Staging environments to be created, one for each Product Owner group, so that each module is tested and errors are isolated. After that, the development team uses the Qualification environment to package a new deliverable with the functional modules and the modules that require a small correction. The remaining modules are later corrected and integrated in the next deliverable, so that the schedule is not delayed.

- Pre-production: this environment is used as the trial before setting the application in production. All the production toolkit, including the batch automatization, schedulers, backups routines, supervision and monitoring is tested with real data. When a new version of a production application is installed, data is exported from the production environment and used in the Pre-production to enable testing with real data. The overall behavior is simulate with real data and with the interaction with other applications.

- Production: this environment supports the application and data for final use. Unlike others environments, the production is the only one to be committed to a Service-Level Agreement (SLA), contracted with the Product Owner.

- Training (optional): is used for users training, should it be required.

2.1.2 Applications Life-Cycle

The previous environments are used in specific phases of the application's life-cycle. These include the specification and analysis, the development, staging, simulation, exploitation,

maintenance and end-of-life.

Specification and Analysis

After the Product Owner expresses new necessities and require a new application, he vouches for the development of a technical solution and the provisioning of the final product (Figure 2.2).

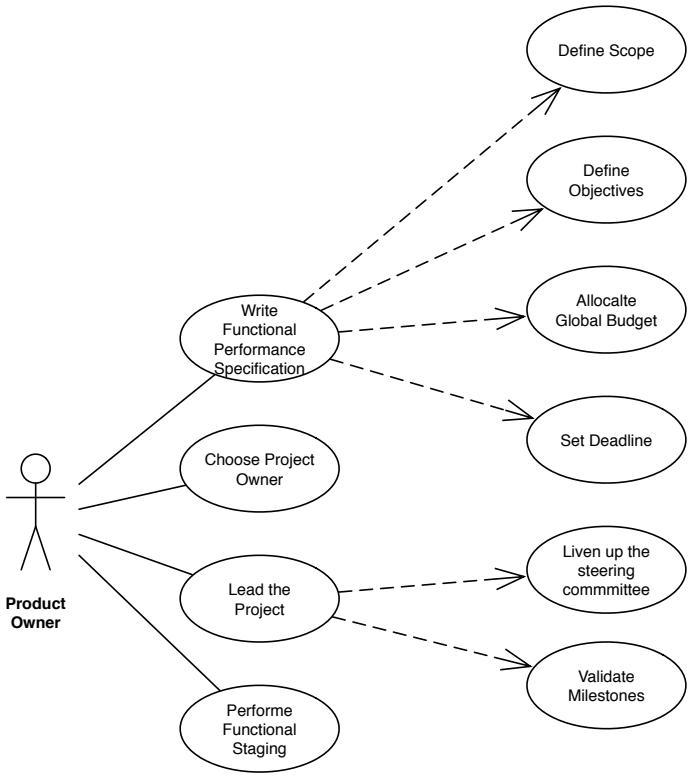

Figure 2.2: Product Owner [3]

A Project Manager is appointed by the Project Owner and, together, they define the contract terms, including the functional performance specification, the technical specification and architecture, followed by the Production domain validation (Figure 2.3).

The validation process starts with the delivery of the Technical Architecture Documentation (TAD) to the Production Architect, which validates the technical choices and ensures compatibility with the components certificated versions, e.g., a specific version of

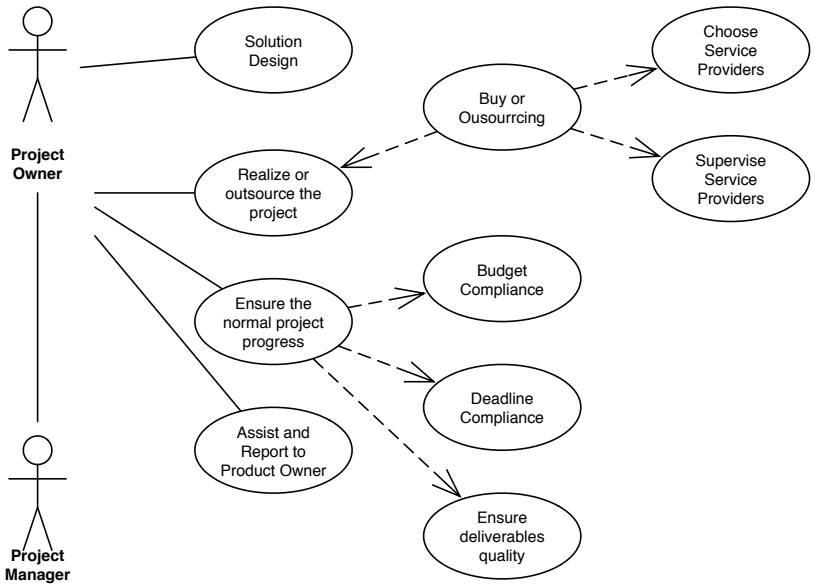

Figure 2.3: Project Owner with Project Manager [3]

a *Tomcat, Mysql, Oracle* to CentOS 7. An example is provided by an European Counsel TAD [7].

The Project Manager is in charge of planning the development, the deployment in the production, the milestones to receive the deliverable and their tests, respecting the deadline and global budget set by the Product Owner (Figure 2.4).

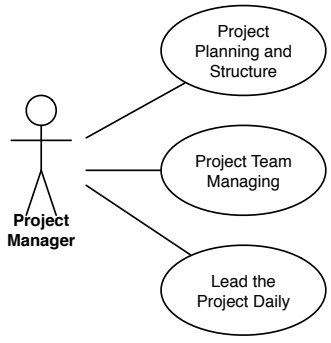

Figure 2.4: Project Manager [3]

Usually, the project is not developed internally, which requires managing an outsourcing agreement. This process is also followed when new functionalities need to be implemented in existent applications.

Development

The Development phase is where the application is developed or new functionalities are added. This will happen in the Development, Integration and/or Qualification environments, which are mainly provided by the Production domain team with just the required application components (Tomcat, Oracle, Jboss, Mysql, Apache or others). The required version is also met, to ensure compatibility with the architecture and the production installation standard.

As described above, the application may be developed internally or it may be outsourced. In both cases, the team has to report to the Project Manager and provide the deliverables (product with documentation of installation and exploration) on time, after unity and integrity testing verification.

Staging

The Staging phase (or Qualification) is composed of two steps: the technical and the functional staging. By definition, staging begins when the Production domain team receive a first deliverable for a new application or an updated version, with associated documentation (TAD, Technical Installation Documentation (TID), Exploitation Documentation (ED)). Then, to meet the deadlines, they need to prepare the staging environment and fulfill the necessary demands of the dependency services.

For each application, it is necessary to make several requests to distinct offices of different domains of the IST department, to build the base line for starting the application qualification. These requests are made based on the TAD, which enables to define the operating system and the resources need, such as RAM, CPU, storage, and the interdependency and their flows between users, components and applications.

Therefore, to create the base line which will enable the Production domain team to start its work, diverse Virtual Machine (VM) or OS-level virtualization (containers) should be ordered to the Exploitation and DNS registration to the Security/Network.

Depending on the scenario, more demands may be needed, for example, if HyperText

Transfer Protocol Secure (HTTPS) connection is required from the Internet, new requests to the Security team are made to open network firewalls ports and to create Secure Sockets Layer (SSL) certificates.

The installation of the architecture component follows, previously defined to support the application (e.g. Tomcat, Oracle, JBoss, MySQL, PostgresSQL, Solr, Apache or others) with the appropriated configurations.

Based on the TID, the steps for correct installation and deployment are followed, including the definition of properties' values, initialization of databases, configuration of batch scripts, and solution artifacts deployment.

Technical tests are made to check correct execution, with logs verification. Moreover, tools are installed and configured to ensure a possible regression in case of errors. These include backup managers, that are configured according to the database used. Finally, tasks are also scheduled for automatic execution.

Then proceeds OAT, which is a common type of non-functional software testing covering key quality attributes of functional stability, portability and reliability (testing installation, compatibility, recovery, regression, security, performance, supervision, …).

After the functional test, the Production domain team helps the Product Owner to conduct the functional tests, using the UAT which describes a process to verify that a solution works for the user and meets all the functional requirement of the contract terms.

The detected errors are reported to the Project Manager, who instructs the development team, to make the corrections in time and furnish a new delivery with updated documentation. Usually, the Production domain receives several deliveries before passing all the required tests, thus it is essential that the Project Manager considers not just the initial tests, but also potential errors, and the time required to correct them, deliver and test. This environment carries out regression testing before the deliverable is installed on the Pre-production and in the Production environments

The procedure is analogous for updating an existing application, with the exception that environments are already available and are functional, hence the preparation before the delivery, is not necessary.

Simulation

After the approval of the technical and functional tests, before deploying into production, the application should be evaluated in a Pre-production environment, specially when testing an updated versions to minimize possible errors.

As the name of this phase suggests, the objective is to simulate an identical environment of the Production environment in the Pre-production. After following the validate procedure to install the approved deliverable in the Staging environment with the updated documentation, the Pre-production is used to test the application's interaction with other components and services. As opposed to the staging, which was used to test the application in an isolate form, the simulation uses real data and tests the interaction and communication with others applications in Pre-production environment. It has the goal to approve the full scenario as describe in the TAD.

To be able to put this environment at the same level of a production platform, the team needs to configure a set of tools to automatize and monitor the environment. The details of those procedures are in the ED, such as information about which processes and partitions are need to be motorized, the CPU and RAM usage limits and the existence of automated centralized backups. Considering the application design and characteristics of each layer, data volume and the operating systems, diverse backup and recovery tools will be chosen. Moreover, the Project Manager defines the job plan to enable Production domain team to configure a job scheduler that orchestrates the real-time business activities, enabling:

- the application execution control;
- limiting direct human intervention to minimize production chain errors;
- respecting dependencies between processes and events in the application or with others;
- to have a common solution for the Production environments to be faster and flexible;
- guaranteeing a certain level of availability in the Production by supporting a recovery plan.

If errors occur, a new delivery will be required and the process goes back to the staging phase.

Production

Once the Pre-production deliverables and the documentation have been validated, the application may be put in production. At this stage, it is supposed to reproduce the simulation steps, now well documented, but in the Production environment. This implies making the needed demands to other services, install the application or an updated version, and configure an automatic and monitoring set of tools, as described above.

If it is the first set in production for the application, it will be necessary to design and try a Disaster Recovery Plan (DRP), define if the availability hours of the application require to have the Production domain team on-call duty, and which type.

The Product Manager and Product Owner acceptance is still required before (re)opening the application to users. They will be in charge to plan when the application will be available with the Production domain team and communicate it to users.

Maintenance

The maintenance is the application daily tasks, including the verification, analysis and modification performed from the moment it is in production until its end of life. Even if an application does not required changes or version updates during several years, it still requires human intervention to stay operational. To keep it operational, the application still need resources, application components who still need to be functional and updated. These may need to be upgraded for security reasons or to keep the compatibility with related application changes.

Obviously, in production, a live application is constantly accessed and data is updated by users, the job scheduler or other automatization. So it requires a daily and constant surveillance, to secure and ensure the integrity and perenniality of data and environment, verifying job scheduler reports and monitoring possible attacks or system failure (overload

and alerts per resources). Furthermore, a great response capacity to analyze and resolve incidents is asked to attend daily demands from other services.

Every configuration request or changes on the application, that do not require a new delivery, is consider as maintenance. However, it is still needed to be tested in Staging and Pre-production environments before being put in production. In case of errors, after analysis, they are reproduced, corrected and tested in the Staging and Pre-production before putting it in production.

For instance, when the batch execution of a job scheduler gives an error because the input file is corrupt, a procedure follows to test and reproduce the error manually in Staging and using the job scheduler in Pre-production. After the cause is pointed and correct, the solution is also tested manually in Staging and uses the job scheduler in Pre-production, so that it is validated applied to the Production environment. Only in case of a new delivery, a correction or a functional addition, the staging and simulation phases are re-done.

End of life

When a decision is made concerning the end of life for an application, the procedure can be seen like a reversed process of putting it in production. After receiving the official confirmation from the Project Manager to decommission the application, the first step is to perform a full backup of the Production environment. Then it is necessary to deactivate all monitoring, automation and backup tasks, after stopping the application services. All the requests made to create the diverse application environments are undone and all the documentation related to the full application's life-cycle (contemplating also the end of life phase), is archived in order to support future reconstruction needs. Finally, the systems of all environments are stopped and all artifacts securely destroyed.

2.2 Challenges

As discussed above, with the exception of analysis, it may look like each phase is represented by one (or more) environments, as the presented environments are the most commonly used for standard applications.

In projects with critical impact, intermediate environments may also be created, such as the use of qualification and diverse staging environments as referred before.

An important aspect to understand regarding the interdependency of teams, is that the Project Owner and the Project Manager are in charge of the development and qualification environment, but Staging, Pre-production and Production are managed by the Production domain team. As a summary, the applications' life-cycle is extensive and requires the collaboration between many people and teams, ensuring the highest quality and continuous operation at all times (Figure 2.5).

Even if the process to put an application in production is defined with coherence (and although it permits to obtain a functional application in production), there may still be some challenges related to coping with the objectives of each team. For example:

- Does it respect the budget, or the defined planning?
- Can any Production domain member recreate (or continue) an instantiation of any environments respecting integrity and consistence?
- Are all the environments of an application synchronized?
- Are teams able to collaborate?
- As Product Owner demands are increasing, can the Production domain team be able to attend them regardless of their own restrictions, and without hiring more people?

With those increasing demands, the Production domain assumes a new set of challenges, requiring to be able to put in production constantly more applications, with various architectures, in less time, maintaining all the environments consistent and without

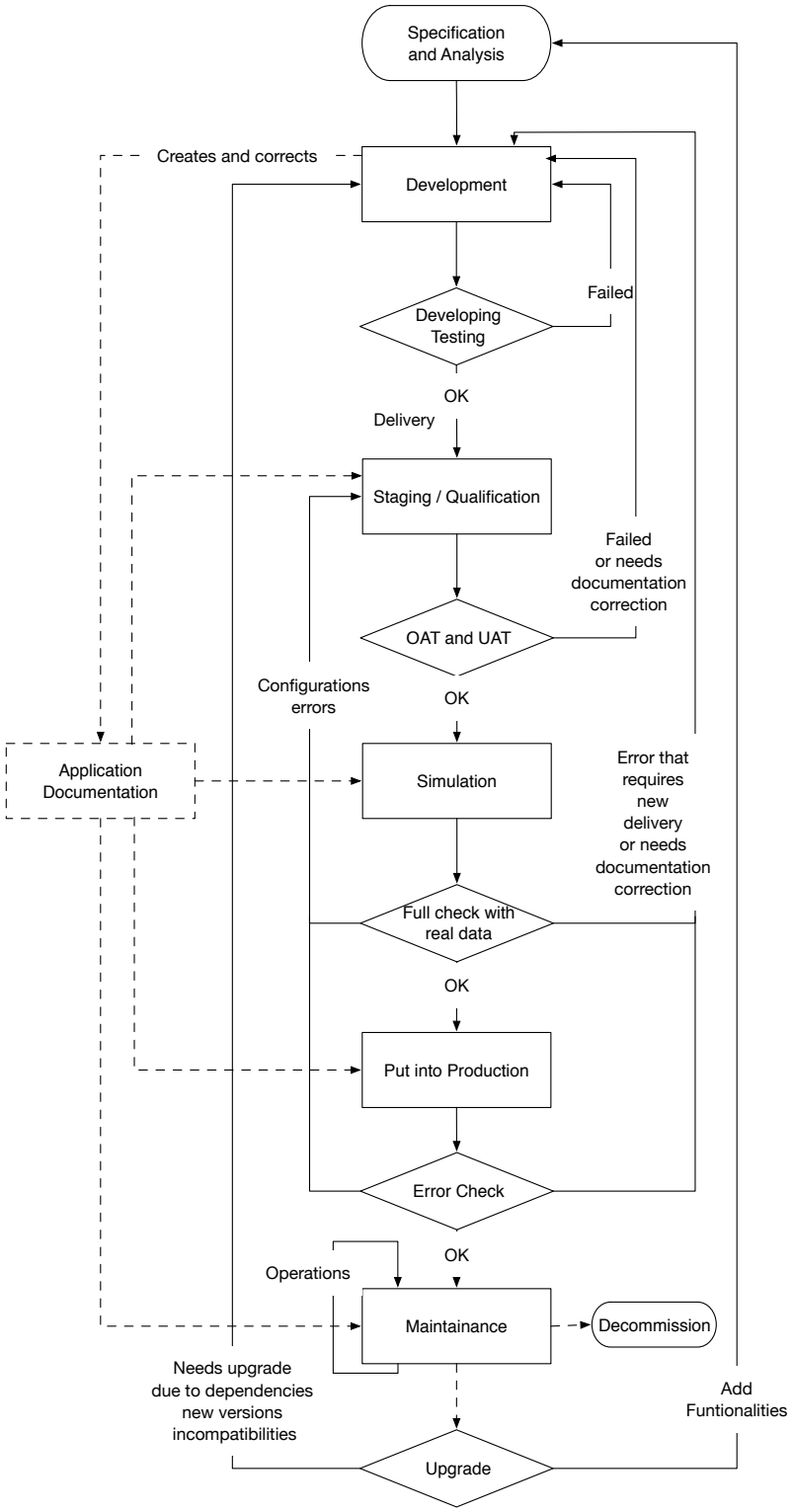

Figure 2.5: Application Life-cycle

adding any new members.

Industrialization of IT has been gaining some attention and may be of help meeting this challenges. It is based on using the innovations and concepts behind the evolution of industrial manufacturing and adapt them to the Production domain activities. This implies a mechanization of production to improve efficiency and cost [9].

2.3 Industrialization

Industrialization comprises the sub-concepts of modularization, standardization, automation and specialization [13]. These are defined and structured with the optimization in mind, maximizing productivity and minimizing cost. The goal of this work is to adopt this principles for the industrialization of the Production domain activities.

It requires defining the Production process and establishing which type of information and data is necessary. This could be done using techniques for modularization, like data flow diagrams or process modeling, and including not just internal processes but also all the people and teams who interact with the Production domain.

All the processes involved in each application stage are defined, and it is only possible to advance to the next step after the previous stage is validated. For example, since the Production architect has to validate the technical specifications, it will not be normally possible for the Project Owner to contract outsourcing or develop an application without it. Missing this process leads, most of the time, to incompatibilities during the technical staging. These incompatibilities require extensive analysis and the development of a new deliverable, that will increase costs and delay the project. More delay can happen if the Project Manager chose to plan all the project without consulting the Production domain schedule, which involves other applications, planned in advance.

Next, it is important to define norms and standards, such as application naming schemes, DNS and servers names, etc, which enables all team members to work in the same way and insuring consistency. Documentation is very important to sustain this.

Each application needs to have diverse types of updated documentation. First, an

application needs an identity card with general identification information: type of backups, production hours, project manager identification, specific services required, stop and restart procedure, and other critical information. Application manuals like TID, TAD, DRP, etc, should be available and updated. The process of requesting service configuration to other services, with the needed information, should be created, to help all services to ensure data and service continuity. Each applications have different type of SLA, specially related with incident response. To help minimize the response time, information should be easily accessible so that even professionals that were not involved in the initial configuration and deployment are able to solve common problems.

Automation is another important aspect. It enables using identical configurations independently of the person that is performing them, which gives consistency. It minimizes the time spent on repeated tasks and makes it easier to respect the standards and norms, enabling reusing previous configurations in other projects. Each application environment will use similar processes, which will allow an easier maintenance of environment synchronization and testing by different actors, in a collaborative way. Moreover, this enables every technical level to understand better the environments, from the developers to the managers, enabling all to validate the different phases.

The relevancy of specialization depends on the enterprise dimension, and/or the applications environments complexity. Hence, depending on the scenario, different type of specialization could be chosen. When the number of applications is high, it is difficult to be able to respect the schedule of different applications, plus incident management. Then it will be more pertinent to specialize the team in two groups: one for application integration and other for incidents management. If applications with big dimension exist, such as SAP ou HRAccess, it would be necessary to dedicate some members to each one. An alternative to dedicate specific members to specific tasks, is to outsource incident management and retain application integration competences within the Production domain team.

2.4 Tools and Technology

2.4.1 Virtualization and Cloud

System virtualization appears as an adequate solution to exploit hardware in a way that multiple operative systems can be installed on the same hardware. Diverse types of solutions are available, including bare metal hypervisors with some services to create and maintain virtual machines. Other possibilities implement virtualization on a host operating system, or operating system level virtualization, that implements containers that can compartmentalize and isolate the execution of services or applications in the same host machine [8].

This type of tools enables backing up the full virtual machine or container, creating several snapshots in order to roll back the operating system to previous state. This functions enable the Production domain team to have more control and to manage the infrastructure in a more integrated way. When using full virtualization, the layer of the hypervisor enables to homogenize the platform where the systems are executed. Therefore each VM can migrate between physical machines, providing failover automatically. Moreover, template images can be use to have base machines pre-configured to be used when needed.

The cloud appears as a way to exploit the use of virtualization in order to provide access to a infrastructure-as-a-service, platform-as-a-service or software-as-a-service. This enables to use all the infrastructure as a pool of resources and to manage the creation of new virtual machines or deployment of services in a more dynamic and automatic way. But the configuration of all systems, services and applications, their provisioning, intra-service orchestration and deployment, takes a lot of time, and is a complex activity. This is a big challenge to the Production domain, considering that they need to maintaining several different environments during the life-cycle of potentially complex applications on large enterprises, which normally have a big IT department, structured in specialized groups.

2.4.2 Configuration Management Frameworks

Dealing with configurations of systems, services and applications is a very time-consuming process that system administrators try to cope with by automating it. They started by using pre-configured images and a set of shell scripts to deploy systems and configurations. It may take the systems to the initial state when needed, but it does not help keeping the system state at the required level, with the necessaries updates and changes. The reuse and share of custom scripts is also very limited. Moreover, this scripts cannot reset the system to a required state [15]. Moreover, manual custom scripts are very prone to human mistakes, difficult to understand, unscalable and unpredictable.

In order to cope with the need of automation of system administration, configuration and management tasks several configuration management frameworks have emerged. Diverse factors push the need of automation, as the difficulty of management of configuration changes across diverse systems, or the necessity of applying configurations on ephemeral cloud-base systems and the ability of reproduce configuration among diverse platforms [14].

Puppet [10] , Chef [6], Salt [12] and Ansible [1] are the most well known technologies is this area.

Puppet is based on a client server architecture where the nodes that will be managed need to install and configure agents (puppets), and normally a central server as the master. The communication and authentication between master and puppets is protected by SSL certificates. The master works as a Certificate Authority and signs the certificate requests of the agents. It supports diverse operating systems, such as Linux, MacOS, other Unix-like OS and Windows. The Puppet configuration is described in manifests, that package the resources and other files. When executed, Puppet compares the configuration of the systems with the one on the manifest and takes the actions needed to change machine state to the one described on the manifest [2]. The configuration is specified in a declarative domain specific language and implemented in Ruby. Puppet has a huge number of modules to enable users to easily describe their configurations.

Chef is similar to Puppet and it is also implemented in Ruby and uses a domain specific language. Is has a central server and agents, which are installed on the machines that will be under Chef management. The authentication between them is made using certificates. It requires a workstation to control and make requests to the central server and to interact with the respective nodes to change configurations. It uses the concept of a cookbook to define the properties and associated values of packages, files, services, templates and users to represent the configuration state [5]. After configurations are made, a command line or a web based solution can be used to associate recipes to nodes and to request execution. Chef supports diverse operating systems such as Linux, MacOS, other Unixes and Windows.

Salt also follows a client server architecture and supports several operating systems, such as Linux, MacOS, other Unixes and Windows. The nodes to be managed need agents (minions) installed in the remote clients. The minions make requests to the master to join. After authorized, the minions are controlled by the master. It supports multiple servers to increase redundancy. A master server can control the slaves, which will control the minions [15]. Minions can make direct requests to master to get more information in other can finish their configuration. It is implemented in Python and uses the YAML language to describe the states of configuration. It has a web interface with few features. Salt operation has several phases. The first is the definition of a state tree and to assign parts of it to servers. Then, minions download the state definitions and compare them to their current state. After that, minions make the changes needed to achieve the desired state and report to the master [4].

Ansible automates configuration management, provisioning, intra-service orchestration, application deployment and many other IT needs [15]. It is agentless and relies on Secure Shell (SSH) for communicating and authenticating. So no other services need to be installed on the nodes that will be managed with Ansible, requiring only the Python execution environment. Ansible gets the necessary hosts from the inventory and executes the code over SSH [4]. Supports idempontency (if a configuration change is applied once, it is not applied a second time, even if Ansible tasks are executed multiple times [14]) and

makes changes noticeable. Ansible was designed to be easy to use and simple to understand. Also, it has an advanced ad-hoc execution engine and is considered to be the 3rd generation configuration management tool [15]. It support diverse modules to facilitate the writing of the playbooks that store the code that represent the tasks to be executed. Ansible supports various operating systems such as Linux, MacOS, other Unix-based OSs and Windows.

Considering all the analyzed Configuration Management Frameworks (CMFs), Ansible seems to be adequate for the industrialization of Production activities. It plays well with other CMFs and with systems managed manually. As the Production domain team deals with diverse types of systems a flexible tool is a requirement. It also is less intrusive since it only needs SSH to remote access and Python available on the server. It can be used to configure systems, deploy applications and orchestrate IT tasks. Finally the learning curve is smaller and the language used to write playbooks corresponds to simple English. This will enable also not so technical staff to understand what the playbook makes.

Chapter 3

Proposal

The industrialization process, as discussed above, relies on the optimization of processes to increase productivity. Essential to this goal is documentation that structures all processes. The documentation should be normalized so that it is easy to develop and read, detailed and exact. In the context of this work, the documentation describe the milestones, deadlines, changes, as well as many other details, and should follow a standard structure that convey requests, application identity cards, documentation of business continuity plans, and others.

Additionally, the existence of naming standards, or rules, accepted by all the organization, reduces the possibilities of misunderstandings and mistakes. Names of servers, applications and services should follow well defined structure so that they are immediately identified and interpreted.

Tasks and processes should also be unequivocally documented, to enable to be followed anytime and independently of who is executing them. Processes need to be supported by the documentation and the tasks and responsibilities of each team should be well defined.

The industrialization of IT application production can be supported by the following:

- Use of ticketing system to control requests and inter-requests between teams;
- Use of request forms to guarantee that all information is correctly provided;
- Use of a name policy to ensure normalization;

- Use of a standard procedure for all new installations or changes;
- Use of repositories with requests support traceability;
- Creation of an application summary sheet to check with requesters if the process should follow;
- Creation of an application sheet that documents all the important information of each configuration;
- Use of automatic processes for normalizing activities;
- Maintain updated documentation of all processes to enable that every team member can be autonomous;
- Define a documentation error or omissions detection, so that the correction can be performed and the application sheet updated.

One important step is the use of automation that can support the normalization of processes, contribute for an updated documentation and support standards and norms, coherence and traceability. This will also facilitate the collaboration and communication between team members and managers.

The following sections presents a framework that can be used in a large enterprise that mimics the organization discussed on Chapter 2.

3.1 Overview

From a generic perspective, the framework's main idea is to reuse configurations and tasks that were previously tested and validated before interacting with the systems in production (Figure 3.1).

The control center uses an Application Programming Interface (API) to request the execution of changes in systems by using a set of tasks and configurations formerly created and tested that uses modules or ad-hoc commands. Although this generic model ensures

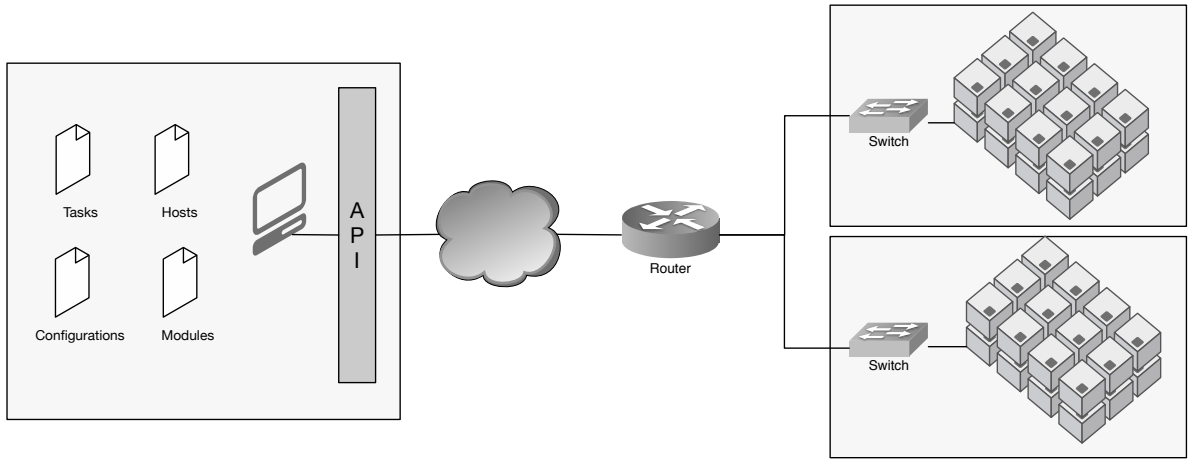

Figure 3.1: Indirect Administration

that the processes are well structured and followed, it does not scale well and it does not deal well with managing all the different tools, scripts and commands necessary by different scenarios and applications.

3.1.1 Version Control Systems

The use of version control systems, such as GIT, allows applying the concept of areas (branches) and updates (versions) that these set of tools suffer during their lifetime, providing coarse grain sets to manage each application.

In other words, each application that needs to be managed will be associated to one GIT repository. Common playbooks and roles will be stored in a shared repository to enable reuse by all actors (Figure 3.2).

Each actor (from the Production or Exploration, for example), will cooperate in the systems configurations. For general configurations and tasks they use the shared repository and then each application can reuse them in their repository for normalization and consistency. When they consider that the configuration is in the required state, Production applies them to systems.

A shared repository is used to maintain general configurations and playbooks that

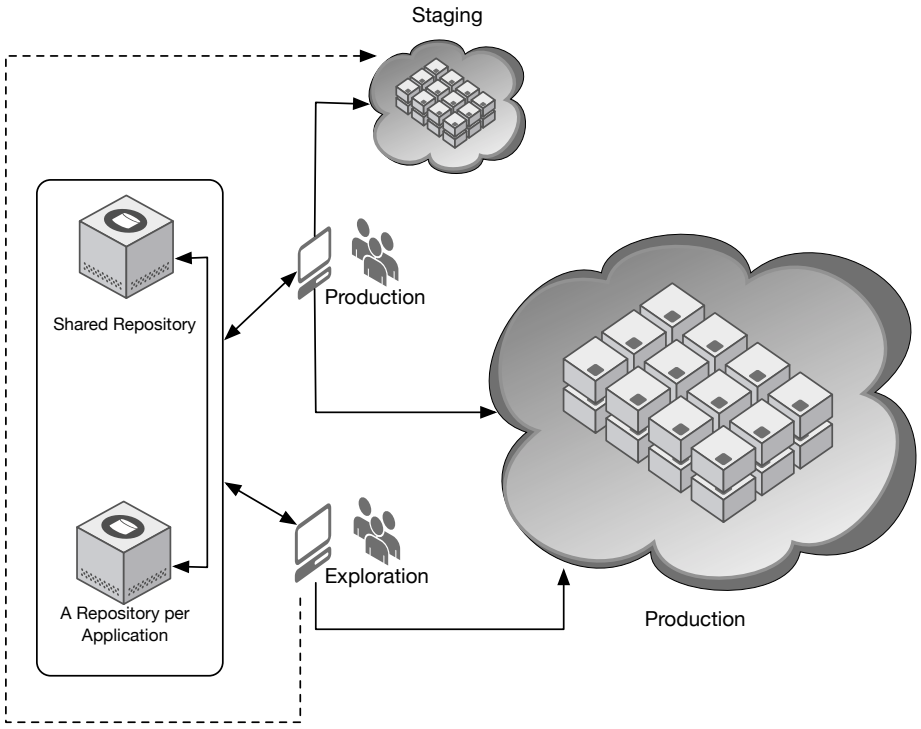

Figure 3.2: Big Picture

can be reused for configuration, deploying, updating and maintaining the different environments per application. The use of the shared repository will enforce the consistency of the diverse systems and normalize the way they are configured. Each application will have its' repository where the shared configurations are reused and instantiated for the required environments.

The use of version control system will also enable the traceability of the changes made to the diverse systems that support the applications, and, when required, old versions can be restored. Actors can view all configuration of the supporting systems on the same repository. Every change to the systems are done by changing the configurations that are on the repository, so its state description can be checked instantly. This enables all actors to understand the big picture and view the configurations that other domains had done, which they may need to continue their tasks.

3.1.2 Systems Configuration Management

In order to support this workflow, a solution is needed to manage the systems configuration. Several CMFs where discussed on Chapter 2 and, considering the requirements, Ansible seems a good option.

It has a short learning curve, it uses SSH and it only depends on the availability of the Python language on the machine that will be administered. It also supports the coexistence with other tools. Ansible can be easily used for configuration, provisioning, deployment, orchestration and execution of hadoc commands and configurations changes for all or just some systems. Such support is very interesting for Production daily activities.

Ansible playbooks are a sequence of tasks, which use built-in or custom modules. There are diverse modules such as `apt` or `yum` to manage package repository for *deb* and *rpm* packages, modules to manage files, execute commands and scripts. More than 500 modules are available to write playbooks or execute them or any other adhoc command.

To execute playbooks the command *ansible-playbook* is used, followed by the playbook name. To execute an ad-hoc command, the command *ansible* is used. Both can use the modules available.

For instance, the playbook from Listing 1 is going to be executed on the webservers group that is defined on the inventory file and needs to elevate privileges to execute the tasks:

- The first task to be executed is install nginx, it requires the module apt and the package named nginx after updating apt caches.
- Then copies (using the module copy) the file *nginx.conf*, that is locally in the relative directory *files/*, to the remote system in the directory */etc/nginx/sites-available/default*.
- After the enabling of configuration is made by linking */etc/nginx/sites-available/default* with the */etc/nginx/sites-enabled/default* using the file module.

```
---
#web-notls.yml
#Author: Reis, Elisete
#Description: Install nginx on webservers running Debian
- name: Configure Webserver with nginx
  hosts: webservers
  sudo: true

  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes

    - name: copy nginx config file
      copy: src=files/nginx.conf dst=/etc/nginx/sites-available/default

    - name: enable configuration
      file: dest=/etc/nginx/sites-enabled/default
↪   src=/etc/nginx/sites-available/default state=link

    - name: copy index.html
      template: src=templates/index.html.j2 dest=/usr/share/nginx/html/index.html
↪   mode=0644

    - name: restart nginx
      service: name=nginx state=restarted
```

Listing 1: Configure Webserver with nginx playbook.

- Then copies a template index.html to the remote server, setting the mode to 644, recurring to the template module.

- At the end the ngix service is restarted by using the service module.

```
ansible webservers -b -m apt -a 'name=vim update_cache=yes'
```

Listing 2: Command for the ad-hoc installation of `vim`.

If the adhoc installation of the *vim* package is required in all the webservers, the command from Listing 2 is enough. The command requests to be executed on the *webservers* group on the inventory. The parameter *-b* requests to raise root privileges to execute the request, *-m apt* loads the module apt, *-a* provide the arguments of the module (the name of the package *vim* with the option *update_cache=yes* to force updating apt cache).

3.1.3 Templates

The Playbooks describe what has to be done to systems to change them to the new state. Using shared playbooks will enforce that all actors follow the same procedure independently of the environment. Playbooks can use a template mechanism for configurations, which, when used, will be instantiated by setting the variables for the environment in question.

Ansible is able to separate the code and data, so that they may be reused. For example, a configuration file specific to an Apache web server usually has the same configuration parameters, such as paths, ports and users, no matter who installs and configures the web server. What changes are the values of the parameters, that can be different based on the environment policy or the hosts where it will be executed. So code is used to configure what is generic, and data what is specific. Ansible uses templates to store the code, written in *Jinja2* [11] and the data is stored in variables. When Ansible needs to use the configuration, it will instantiate the resulting file dynamically by using the information on the template and on the variables (Figure 3.3). The templates can use variables or facts that are automatically discovered by Ansible or registered when executing tasks. The most common facts are hostname, ip addresses, system architecture, operating system, processor use and memory.

Variables can be defined at various levels, related to the playbook, to the role, or to the hosts where it will be configured. If the definition of variables respects the policy of names, all systems will be normalized and consistent.

3.1.4 Roles

Ansible enables code reusing, allowing to include playbooks inside playbooks. Also, it enables to aggregate playbooks and configurations, combining them in a clean way with an abstraction that represents a role. When the same role is needed it can be easily reused.

Roles are defined in a directory structure, grouping all the related concepts.

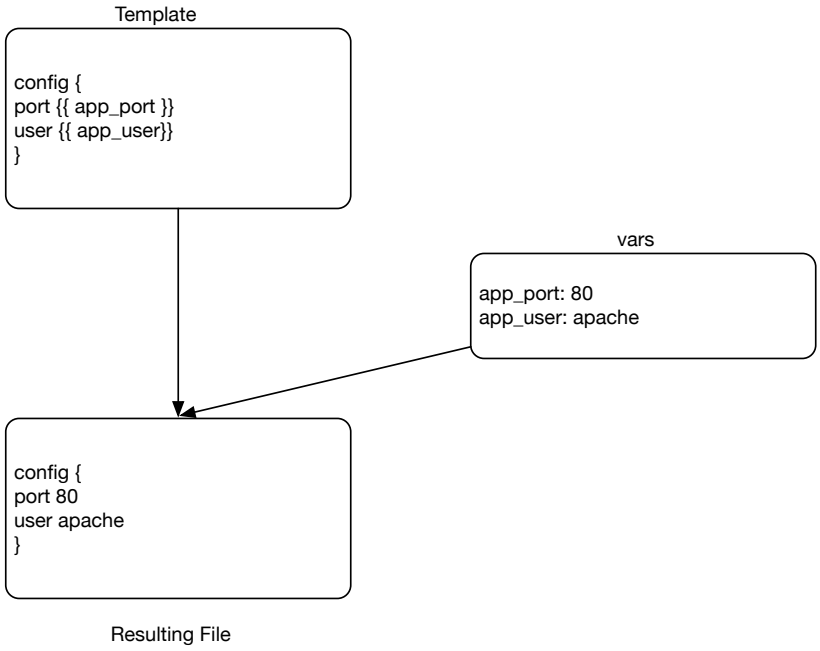

Figure 3.3: Templating

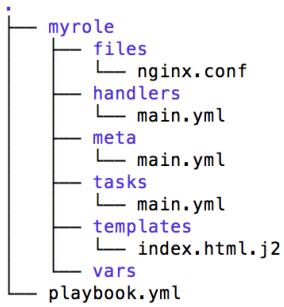

Figure 3.4: Example role structure

The directory *myrole* of Figure 3.4 contains the structure of the role and includes the following sub-directories:

- files: contains regular files that the role may transfer to remotes machines;
- handlers: all handlers are implemented here;
- meta: contains references to dependencies and other information;
- templates: stores the files with the templates used on the role;

- tasks: have all the tasks that normally reside on the playbook. Can reference files and templates without using the path;
- vars: variables that can be used in configurations are specified on this directory.

Ansible will search for and load the *main.yml* file on the *handlers, meta and tasks* directories. If more files are used to divide the implementation they can be included on the main files.

Considering this division for the role structure, the previous *nginx* installation example playbook can be change to the one of Listing 3. The tasks, paths to the files and templates are not needed to specify since Ansible will search them on the files and templates directory.

```
---
- name: install ngix
  apt: name=nginx update_cache=yes

- name: copy nginx config file
  copy: src=nginx.conf dst=/etc/nginx/sites-available/default

- name: enable configuration
  file: dest=/etc/nginx/sites-enabled/default src=/etc/nginx/sites-available/default
 ↪  state=link
  notify:
    - restart nginx

- name: copy index.html
  template: src=index.html.j2 dest=/usr/share/nginx/html/index.html mode=0644
```

Listing 3: Example Nginx Role Tasks.

Some handlers can be executed if notified on the tasks file (Listing 4).

```
---
- name: restart nginx
  service: name=nginx state=restarted
```

Listing 4: Example Nginx Role Handlers.

The Listing 5 shows and example of role dependency.

```
---
dependencies:
  - { role: xpto }
```

Listing 5: Example Nginx Role Meta.

The file *playbook.yml* on the top directory is where the role is called (Listing 6), describing the states, the hosts and the roles to be used.

```
---
- hosts: webservers
  roles:
    - role: myrole
```

Listing 6: Example Playbook calling Nginx Role.

3.1.5 Automatic Documentation

According to the requirements described in the documentation, the playbooks are declarative and written in Ain't Markup Language (YAML) which is easily understandable by non technical staff. A parser is used to dynamically generate a draft documentation from the playbooks and roles on repositories. Based on the names that each task is given in YAML files it is possible to extract them in order to explain what composes the playbook. Even if it uses roles, this can be achieved by using *ansible-playbook* with the argument *–list-tasks* (Figure 3.5).

```
playbook: playbooks/web-notls.yml

  play #1 (webservers): Configure Webserver with ngix   TAGS: []
    tasks:
      install ngix      TAGS: []
      copy nginx config file    TAGS: []
      enable configuration      TAGS: []
      copy index.html   TAGS: []
      restart nginx     TAGS: []
```

Figure 3.5: Tasks executed by example playbook on Listing 1.

Figure 3.6 shows the result for the example role describe on the structure presented on Figure 3.4.

```
playbook: playbooks/web-notls.yml

  play #1 (webservers): Configure Webserver with ngix   TAGS: []
    tasks:
      install ngix      TAGS: []
      copy nginx config file    TAGS: []
      enable configuration      TAGS: []
      copy index.html   TAGS: []
      restart nginx     TAGS: []
```

Figure 3.6: Tasks executed by example role

This can be used to rapidly verify and discuss each playbook and roles at functional level with all the involved actors.

When listing the tasks it also shows information about tags for each task. The keyword *tag* can be use to associate one or more tags to each task on a playbook. Therefore, when requesting the execution of a playbook, the user can include the tasks that only have some tags and with that approach execute only the tagged tasks.

A tool called *ansible-docgen* is also available, that generates documentation for annotated playbooks and roles. It creates a markdown file with basic information of the playbooks and roles that can be used as a base for documentation production (Listing 7).

```
## Playbook: [playbooks/web-notls.yml](playbooks/web-notls.yml)
> **Author:** Reis, Elisete

> **Description:** Installs nginx on webservers running Debian

> **Task:** install ngix

> **Task:** copy nginx config file

> **Task:** enable configuration

> **Task:** copy index.html

> **Task:** restart nginx


Generated by [ansible-docgen](https://www.github.com/starboarder2001/ansible-docgen)
```

Listing 7: Automated Documentation for example playbook

This generated document can be stored in the same repository that stores the playbook

and the roles. Users can also cooperate to improve and maintain the documentation updated. From this markdown files, Hypertext Markup Language (HTML) pages or Portable Document Format (PDF) files can be generated.

3.1.6 System Modification/Changes

Other aspect that the Production domain team often needs to normalize is the process of introducing changes to the systems. When changes need to happen, some times it is necessary to execute dependent tasks in some other system before applying the changes and other tasks after these as well.

For a better understanding, consider a load balancer and various web servers that support a web application. To update the web servers without downtime, they will be updated one by one. The following procedure should be followed for each node:

- Disable the host on the load balancer;
- Run the roles needed to make the update and changes needed;
- Wait for the host and service to be running and listening;
- Register back the host on the load balancer.

In order to write a playbook that respects the previous workflow, *pre_tasks:* and *post_tasks:* are used to describe the tasks to be executed before and after the changes are applied (Listings 8).

The hosts that execute this playbook belong to the *webservers* group on the inventory. The *Serial* keyword is used to define the number of hosts that will execute the tasks each time. This way the update can happen without downtime. As there are few hosts, the update may be done one at a time.

Before executing the roles, the *pre_tasks* is executed. In this case the host is disabled in the load balancer. The module haproxy is used to request disabling the host in all the load balancers. The keyword *delegate_to* request the task to be executed in other host. In this case, it corresponds to all items that are on the group loadbalancer on the inventory.

```
---
#This playbook updates webservers on by one

- hosts: webservers
  serial: 1

  # Execute before updating:
  pre_tasks:
  - name: Disable the host on load balancer
    haproxy: 'state=disabled backend=mysiteapp host={{ inventory_hostname }}
↪   socket=/var/lib/haproxy/stats'
    delegate_to: "{{ item }}"
    with_items: groups.loadbalancer

  roles:
  - common
  - nginx
  - php5-fpm
  - mysite

  # Run after:
  post_tasks:
  - name: Wait for the host and service to be running and listening
    wait_for: 'host={{ ansible_default_ipv4.address }} port=80 state=started
↪   timeout=80'

  - name: Register back the host on the load balancer
    haproxy: 'state=enabled backend=mysiteapp host={{ inventory_hostname }}
↪   socket=/var/lib/haproxy/stats'
    delegate_to: "{{ item }}"
    with_items: groups.loadbalancer
```

Listing 8: Workflow for updating without downtime, requiring the roles common, nginx, php5-fpm, mysite.

The roles common, nginx, php5-fpm and mysite are executed in one web server.

Next, the *post_tasks* will wait for the host to be started and listening on port 80. Finally, the host is enabled on all load balancers, and start receiving requests.

3.2 Usage Scenario

The main idea is that the configuration and the tasks to change state are executed indirectly on the systems by first creating the playbooks and the roles needed, testing them

and, only after, executed on the production systems.

3.2.1 Common Repositories

The first step is to define common repository with general roles and playbooks for the Production domain tasks. After approved, these can be reused in the configurations of the applications that will have their own repository. For better management, the shared repository should have, in fact, a directory for the playbooks and one per role (Figure 3.7).

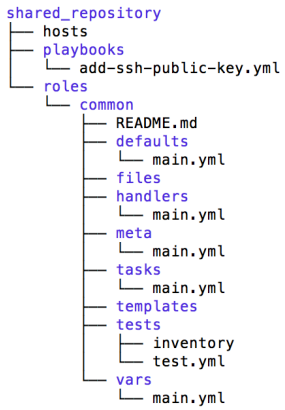

Figure 3.7: Shared Repository Layout

The hosts file has the inventory with all machines that compose the various environments managed by the Production domain team.

The directory *playbooks* can contain playbooks for common tasks or procedures that can be reused in daily activities. Special playbooks can be created to support patching a set of servers, or to check some configurations when some problem is detected or a security patch is available to mitigate vulnerabilities.

The *roles* directory will aggregate all the tasks and configurations needed to implement some functionality on the systems. The directory structure is similar to the one explained previously, with the addition of a directory called *defaults* that stores the default variable values.

The idea is that roles and playbooks will be created, tested and checked if they respect the normalization and consistency requirements. After, they can be used on production

environment.

The roles that should be created, tested and deployed in the production systems are the ones that support common configurations that should be transversal to all nodes.

For example, a role with the base configuration and the checking procedures for all systems, with minimal package installation, configuration of remote administration services, initial firewall and system update is one of the firsts.

Also, a role for monitoring should be made available since all machines will need to be monitored.

Some roles will depended on others and some will also depend on some applications. So the order of creation needs to respect those dependencies. Planing should be done in order to follow a good sequence in creating the base roles and to deploy of applications that will support others.

3.2.2 Application Repositories

For each application on Production, a repository should be created in other to be used to manage it during all phases on the diverse environments. Figure 3.8 depicts the possible layout for an application repository that needs the roles *common*, *web* and *database*.

This layout is good when the environments have similar configurations and therefore the value of the variables is almost the same.

The files *production* and *staging* store information of the host inventory of the systems to be used on production and the staging environment. When executing the playbook, the administrator only needs to change which inventory file he is using to target the staging or the production environment. The directory *group_vars* and *host_vars* stores files with variable values for some groups or to specific hosts.

The *master.yml* file includes the roles files (*common.yml*, *database.yml*, *web.yml*), Each role file defines how each role is going to be configured.

When a big difference between the configuration values on the diverse environments exists, it is better to separate inventories in different directories and, inside them, to have

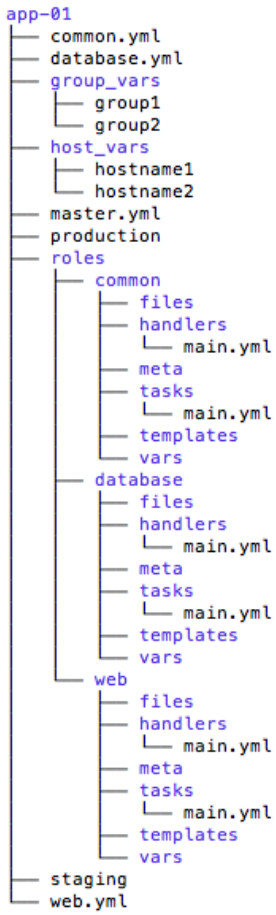

Figure 3.8: Application Repository Layout

the directories with the variables per group and per hosts (Figure 3.9).

On this approach, each environment has its directory inside the directory inventories. The environment directory aggregates its *group_vars* and *host_vars*. This will minimize errors in the associations of the variables values for each environment since on this files only the values for a specific environment exist.

As we can see, the infrastructure will be represented by the code on the playbooks, roles and configurations. In order to maintain the infrastructure state we will use some workflows that normally developers use to manage their code. When new services need to be created, tested and deployed a fork of the repository that represents the state is made. A development environment is created with the same configurations, different branches can be created for each new thing to be tested or per team working in the configuration.

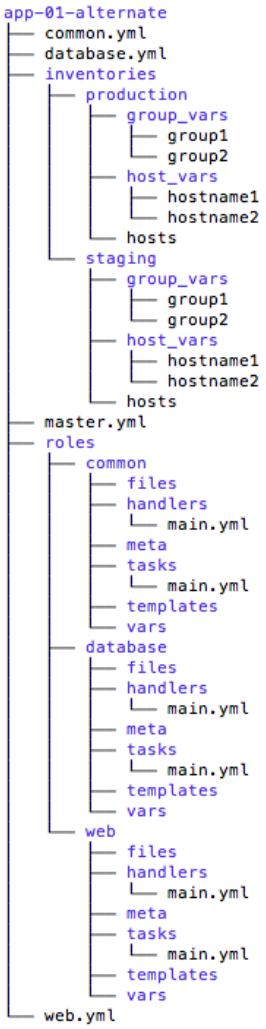

Figure 3.9: Alternate Application Repository Layout

After all changes are made, branches are merged and the full playbook executed against the staging environment. After all the tests and green light for production, the fork is pulled to the original and then executed using the production inventory to make the changes need to update to the new state.

3.3 Workflows

3.3.1 New Role or Playbook

Playbooks and roles can be reused in order to maintain consistency and achieve normalization. Therefore, when a task needs to create a new reusable artifact, a specific sequence of steps are followed (Figure 3.10).

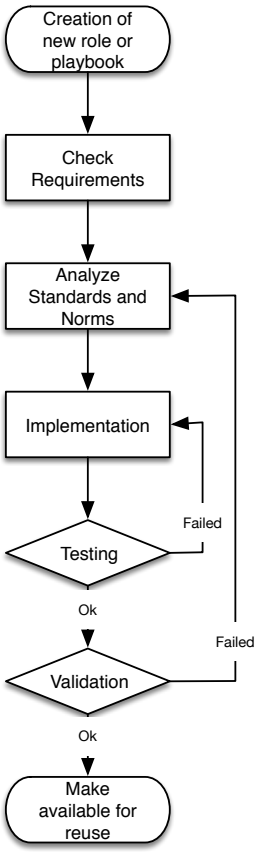

Figure 3.10: Creation of a shared playbook or role

The first step is to analyze the requirements, followed by the verification of the norms and standards that need to be respected. If it is a new role, a new repository is created and, if it is a simple playbook, a branch should be created in the common repository for the new playbook. After the implementation is tested, if it fails, go back to reimplementation. The validation phase is where the implementation is checked. If it does not respect the

norms and standards, it goes back to the analysis phase. At the end, the new artifact, should be made available so they can be used by the Production. In the case it is a simple playbook, the branch should be merged with the master. Otherwise, the new repository should be shared.

3.3.2 New Application

Other important workflow is the one that supports a new application in a infrastructure (Figure 3.11).

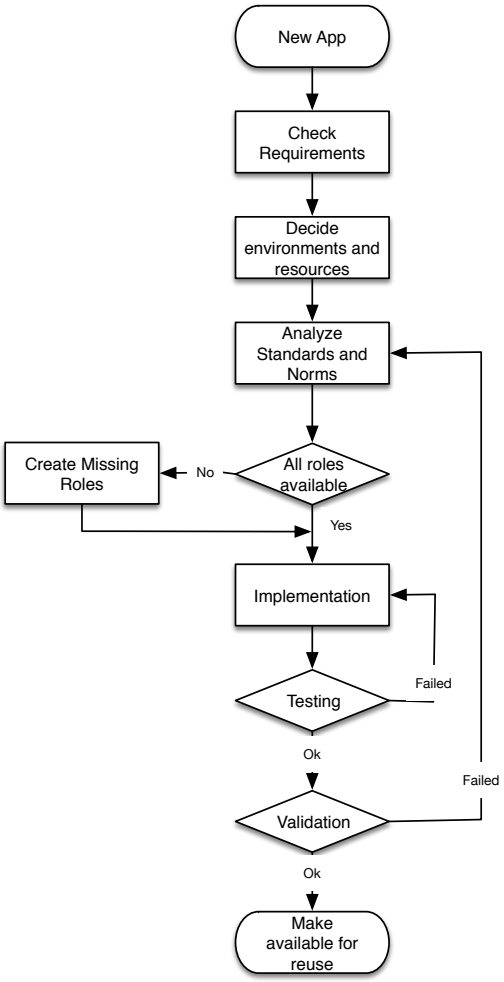

Figure 3.11: Supporting a new application

When the support for a new application is needed, a repository is created to store

all the configuration. Requirements are studied and environments that would be used by the application are composed. The analysis of the norms and standards that should be respected follows, and also the verification that all the roles that compose the baseline of the applications are available. If new roles need to be developed, the process depicted on Figure 3.10 should be followed, in order to be able to use them. Next follows the implementation phase where the roles are reused and configurations are programmed to support the application and its deployment. The implementation is tested in the testing environment and if it fails goes back to reimplementation. If it passes, goes to the validation phase to check norms and standards. Failing the validation implies going back to the analysis phase, otherwise, the repository should be shared to enable use by all Production actors.

3.3.3 Application and System Update

One usual activity is the updating applications and systems. Considering an update of an application already in Production, the workflow on Figure 3.12 should be followed.

The first step is to check the requirements of the update, the standards and the norms. Then, follows cloning the repository of the application and creating the branches to the new functionalities or changes. In each branch, changes are implemented and tested. Failures will take back to reimplementation if it is a functionality or implementation problem, or to reanalyzing standards and norms if is a validation failure. When all tests succeed, each branch is merged with the master, and they are executed in the staging environment to enable others to test the updates. After the final validation, the forked repository is pushed to the original and executed on the production environment.

This approach can also be used to check if system and underline packages can be updated without problems. The following process can be used:

- Update the roles for new versions;
- Test and make them available to reuse;

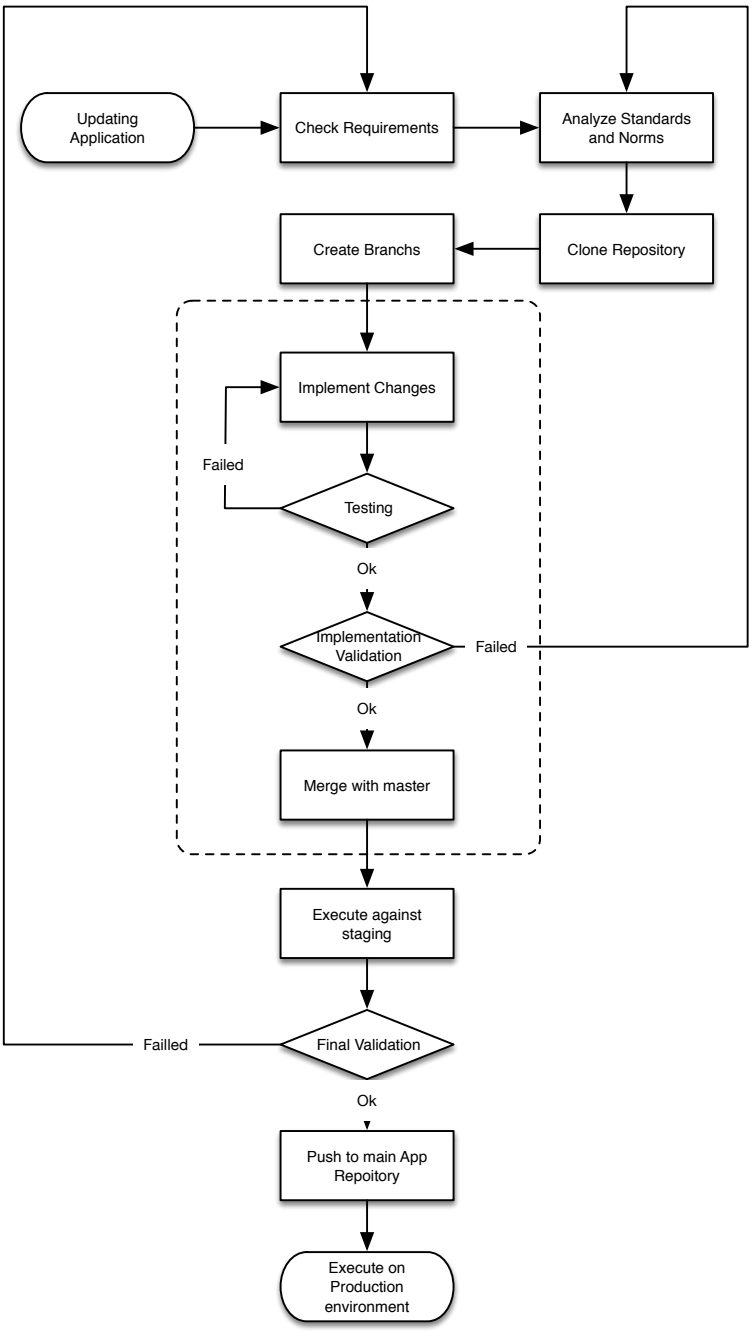

Figure 3.12: Updating Application in Production

- Clone the application repository;

- Import the new roles;

- Test it on testing environment;
- Merge the forked branch with the forked master;
- If success and authorized proceed to stagging;
- And after testing and green light;
- Push the forked branch to application repository;
- Execute it against production environment.

The previously workflows are the base of the proposed framework, where the interaction with system is made indirectly by playbooks and roles when possible.

The idea is to reuse them in order to minimize error, maintain consistency and respect standards and norms.

Chapter 4

Testing and Analysis

Previously, the foundations for the industrialization of the production tasks were laid. The current chapter establishes a proof-of-concept scenario, which includes the life-cycle of a single application considering different actors collaborating.

4.1 Test Scenario

The test environment mimics a real production scenario, composed of a staging area, in which the applications are tried and configured before moving to the production area, where the applications are made available for wide use. Each area requires a set of machines, one for the staging and the other for the production areas (Figure 4.1).

A GIT repository was additionally added, to support storing several versions of the configuration data required by the proposal previously described, as well as to support the collaboration between different people and domains.

The test scenario was built with two VMware ESXi nodes to execute all the required virtual machines. One node executed the staging machines and the other the production ones. The GIT repository was installed in an independent virtual machine, executed in the ESXi also responsible for the production node.

The application that was used as an example provides a dynamic web service with high availability and high capacity. The information is stored in a relational database

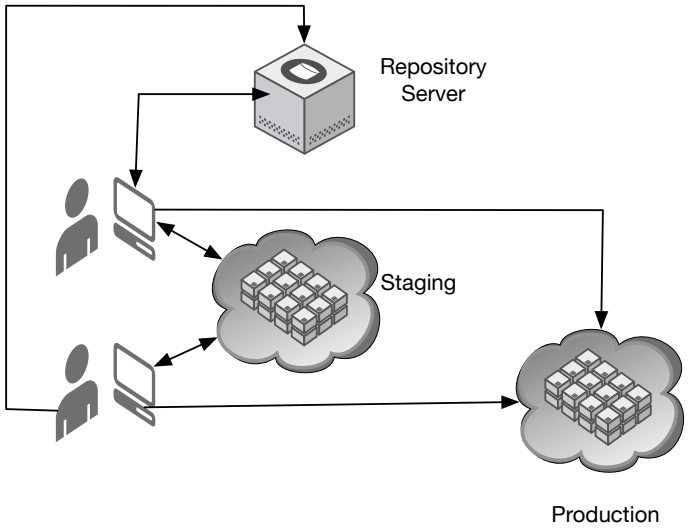

Figure 4.1: Test scenario

and the service is provided by two web servers, in a high availability configuration: load balancer, web server 1 and web server 2 (Figure 4.2).

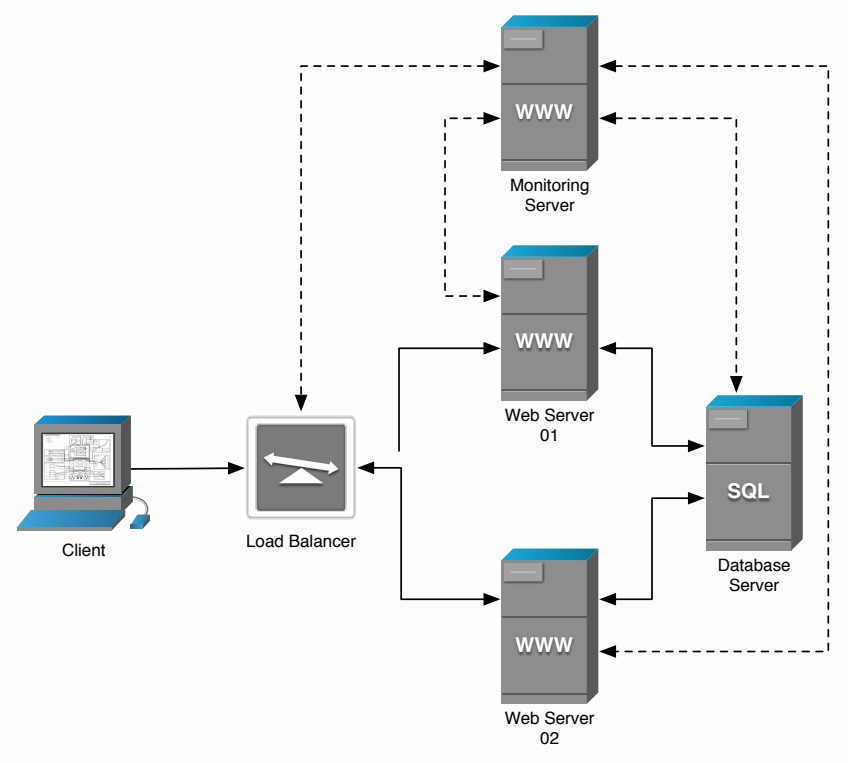

Figure 4.2: Architecture of the example application

A load balancer splits the load between two web servers that can access the database server. In addition, the systems' state is monitored with an additional service, also web based running in a Monitoring server.

All the servers are under Linux CentOS 7 with SELinux in enforce mode. Some examples of Ansible roles and playbooks were used has a base to our tests.

4.2 Test roles

Considering the architecture of the application that we want to test, the following roles where decided to be implemented:

- A common role to configure the base machine that supports every component;
- A role to configure the database server;
- A role to configure a base of the web servers tier;
- A role to configure the load balancer between the web servers;
- A role to deploy the web application;
- A role to deal with the monitoring of the components (using Nagios).

Following the structure used to define a role (Figure 3.4) a repository per role was created.

As the staging and production areas will be running CentOS 7, the actors developing and testing the roles also need to have same version installed. Vagrant (https://www.vagrantup.com) was used to assist the creation, management and destruction of machines, to test the roles and playbooks.

After the functional tests and validations, the roles were made available in the GIT repository for future use.

4.3 Bootstrapping a new application

After testing and validating the roles, they are available for bootstrapping a new application. The followed structure is similar to the one previously discussed (Figure 3.9), which creates different directories per environment to aggregate the inventory and their variables. In this context, a repository named *ap01* was created to store this file structure.

Diverse users work in different branches and, in the end, merge and test in the staging area. The command *ansible-playbook -i inventories/staging/hosts master.yml* may be used to check the tasks that will be executed (Listings 9).

```
ansible-playbook -i inventories/staging/hosts master.yml --list-tasks

playbook: master.yml

  play #1 (all): all        TAGS: []
    tasks:
      common : Install python bindings for SE Linux        TAGS: []
      common : Install EPEL repo        TAGS: []
      common : install some useful nagios plugins        TAGS: []
      common : Install ntp        TAGS: [ntp]
      common : Configure ntp file        TAGS: [ntp]
      common : Start the ntp service        TAGS: [ntp]
      common : insert iptables template        TAGS: []
      common : test to see if selinux is running        TAGS: []

  play #2 (dbservers): dbservers        TAGS: [db]
    tasks:
      db : Install MariaDB package        TAGS: [db]
      db : Configure SELinux to start mysql on any port        TAGS: [db]
      db : Ensure mariadb is running and starts on boot        TAGS: [db]
      db : Change root user password on first run        TAGS: [db]
      db : Create Mysql configuration file        TAGS: [db]
      db : Ensure the anonymous mysql user ""@{{ansible_hostname}} is
↪  deleted        TAGS: [db]
```

```
    db : Ensure the anonymous mysql user ""@localhost is deleted        TAGS: [db]
    db : Ensure the mysql test database is deleted        TAGS: [db]
    db : Create Application Database        TAGS: [db]
    db : Create Application DB User        TAGS: [db]

  play #3 (webservers): webservers        TAGS: [web]
    tasks:
    base-apache : Install http        TAGS: [web]
    base-apache : Configure SELinux to allow httpd to connect to remote
↪   database        TAGS: [web]
    base-apache : http service state        TAGS: [web]
    web : Copy the code from repository        TAGS: [web]

  play #4 (lbservers): lbservers        TAGS: [lb]
    tasks:
    haproxy : Download and install haproxy        TAGS: [lb]
    haproxy : Configure SELinux to let haproxy connet to any        TAGS: [lb]
    haproxy : Configure the haproxy cnf file with hosts        TAGS: [lb]
    haproxy : Start the haproxy service        TAGS: [lb]

  play #5 (monitoring): monitoring        TAGS: [monitoring]
    tasks:
    base-apache : Install http        TAGS: [monitoring]
    base-apache : Configure SELinux to allow httpd to connect to remote
↪   database        TAGS: [monitoring]
    base-apache : http service state        TAGS: [monitoring]
    nagios : install nagios        TAGS: [monitoring]
    nagios : Apply SELinux boolean nagios_run_sudo        TAGS: [monitoring]
    nagios : Apply SELinux boolean logging_syslogd_run_nagios_plugins        TAGS:
↪   [monitoring]
    nagios : Create nagios socket tmp files        TAGS: [monitoring]
    nagios : Check/fix systemd service file        TAGS: [monitoring]
    nagios : Copy SE policy for nagios        TAGS: [monitoring]
    nagios : Install SE policy        TAGS: [monitoring]
    nagios : create nagios config dir        TAGS: [monitoring]
```

```
nagios : configure nagios        TAGS: [monitoring]
nagios : configure localhost monitoring        TAGS: [monitoring]
nagios : configure nagios services        TAGS: [monitoring]
nagios : create the nagios object files        TAGS: [monitoring]
nagios : start nagios        TAGS: [monitoring]
```

Listing 9: Tasks to be executed on staging

The evaluation and test is performed on the staging area. If all is checked and working as expected, the playbook is executed on the production area through the command *ansible-playbook -i inventories/production/hosts master.yml*

To verify that everything executed without errors the monitoring server is checked, to confirm that every component is running (Figure 4.3).

Host	Service	Status	Last Check	Duration	Attempt	Status Information
db1	Current Load	OK	10-24-2016 15:17:33	0d 0h 12m 16s	1/4	OK - load average: 0.00, 0.03, 0.05
	Current Users	OK	10-24-2016 15:18:31	0d 0h 11m 18s	1/4	USERS OK - 0 users currently logged in
	Swap Usage	OK	10-24-2016 15:19:29	0d 0h 10m 20s	1/4	SWAP OK - 100% free (2047 MB out of 2047 MB)
	Total Processes	WARNING	10-24-2016 15:18:26	0d 0h 9m 23s	4/4	PROCS WARNING: 363 processes with STATE = RSZDT
lb1	Current Load	OK	10-24-2016 15:16:24	0d 0h 8m 25s	1/4	OK - load average: 0.00, 0.03, 0.05
	Current Users	OK	10-24-2016 15:15:41	0d 0h 9m 8s	1/4	USERS OK - 0 users currently logged in
	HAProxy Load Balancer	OK	10-24-2016 15:17:45	0d 0h 12m 4s	1/4	HTTP OK: HTTP/1.1 200 OK - 350 bytes in 0.004 second response time
	Swap Usage	OK	10-24-2016 15:18:42	0d 0h 11m 7s	1/4	SWAP OK - 100% free (2047 MB out of 2047 MB)
	Total Processes	WARNING	10-24-2016 15:17:40	0d 0h 10m 9s	4/4	PROCS WARNING: 363 processes with STATE = RSZDT
localhost	Current Load	OK	10-24-2016 15:15:38	0d 0h 9m 11s	1/4	OK - load average: 0.00, 0.04, 0.05
	Current Users	OK	10-24-2016 15:16:36	0d 0h 8m 13s	1/4	USERS OK - 0 users currently logged in
	PING	OK	10-24-2016 15:18:13	0d 0h 11m 36s	1/4	PING OK - Packet loss = 0%, RTA = 0.06 ms
	Root Partition	OK	10-24-2016 15:17:56	0d 0h 11m 53s	1/4	DISK OK - free space: / 16791 MB (93% inode=99%):
	SSH	OK	10-24-2016 15:18:54	0d 0h 10m 55s	1/4	SSH OK - OpenSSH_6.6.1 (protocol 2.0)
	Swap Usage	OK	10-24-2016 15:14:52	0d 0h 9m 57s	1/4	SWAP OK - 100% free (2047 MB out of 2047 MB)
	Total Processes	WARNING	10-24-2016 15:18:49	0d 0h 9m 0s	4/4	PROCS WARNING: 363 processes with STATE = RSZDT
web1	Current Load	OK	10-24-2016 15:19:14	0d 0h 10m 35s	1/4	OK - load average: 0.00, 0.02, 0.05
	Current Users	OK	10-24-2016 15:15:48	0d 0h 9m 1s	1/4	USERS OK - 0 users currently logged in
	Swap Usage	OK	10-24-2016 15:18:08	0d 0h 11m 41s	1/4	SWAP OK - 100% free (2047 MB out of 2047 MB)
	Total Processes	WARNING	10-24-2016 15:17:06	0d 0h 10m 43s	4/4	PROCS WARNING: 363 processes with STATE = RSZDT
	webserver	OK	10-24-2016 15:15:03	0d 0h 9m 46s	1/4	HTTP OK: HTTP/1.1 200 OK - 350 bytes in 0.003 second response time
web2	Current Load	OK	10-24-2016 15:16:01	0d 0h 8m 48s	1/4	OK - load average: 0.00, 0.04, 0.05
	Current Users	OK	10-24-2016 15:17:46	0d 0h 12m 3s	1/4	USERS OK - 0 users currently logged in
	Swap Usage	OK	10-24-2016 15:19:01	0d 0h 10m 48s	1/4	SWAP OK - 100% free (2047 MB out of 2047 MB)
	Total Processes	WARNING	10-24-2016 15:16:19	0d 0h 11m 30s	4/4	PROCS WARNING: 363 processes with STATE = RSZDT
	webserver	OK	10-24-2016 15:19:17	0d 0h 10m 32s	1/4	HTTP OK: HTTP/1.1 200 OK - 350 bytes in 0.004 second response time

Figure 4.3: Nagios

4.4 Common operations

Some activities are meant to be executed several times. The associated commands can be repeated with the command *ansible* and its modules. This enables the administrators to execute the same command in all machines or in a specific group of machines.

For example, to check the kernel version and type that is running on each machine, (and

since there is no module available for this purpose) it is necessary to create a recurrent command that executes the `uname -a` command. For that, it is necessary to issue the command *ansible -i inventories/production/hosts all -m shell -a 'uname -a'*.
Other activities can be performed by developing other playbooks. As an example, a playbook to backup all databases on the database server was implemented and registered.

4.5 Composed operations

Sometimes, it is necessary to implement changes that affect different components. Moreover, some processes need to be respected when applying those changes.
For instance, consider the update of web servers minimizing the downtime of the service. That may be done by updating one web server at a time (Listings 10). So for each web server, first disable the alerts on the Nagios platform and remove the server from the load balancing (haproxy). During the downtime, Nagios will not generate alerts and the load balancer will not try to use the server that is being updated. Then follows the execution of the common base-apache and web playbooks. If everything goes without an error, it waits for the server to start listening again in port 80, and then re-enables the node in the load balancer and re-activates the Nagios alerts to the node.

```
---
#updatingtest.yml
#Author: Reis, Elisete
#Description: Updates all webserver one by one. Executes the common, base-apache and
 ↪  web.roles.
#Configurations and packages updates are made.
#
#Gather all facts for all host to be able to use variables needed in all tasks
- hosts: all
  tasks: []

- hosts: webservers
  serial: 1
```

```
  # Rub before update
  pre_tasks:
  - name: disable nagios alerts for this host webserver service
    nagios: 'action=disable_alerts host={{ inventory_hostname }} services=webserver'
    delegate_to: "{{ item }}"
    with_items: groups.monitoring

  - name: disable the server in haproxy
    haproxy: 'state=disabled backend=myapplb host={{ inventory_hostname }}
↪   socket=/var/lib/haproxy/stats'
    delegate_to: "{{ item }}"
    with_items: groups.lbservers

  roles:
  - common
  - base-apache
  - web

  # Run after
  post_tasks:
  - name: wait for webserver to come up
    wait_for: 'host={{ ansible_default_ipv4.address }} port=80 timeout=80'

  - name: enable the server in haproxy
    haproxy: 'state=enabled backend=myapplb host={{ inventory_hostname }}
↪   socket=/var/lib/haproxy/stats'
    delegate_to: "{{ item }}"
    with_items: groups.lbservers

  - name: re-enable nagios alerts
    nagios: 'action=enable_alerts host={{ inventory_hostname }} services=webserver'
    delegate_to: "{{ item }}"
    with_items: groups.monitoring
```

Listing 10: Updating Minimizing Downtime

4.6 Discussion

All developers and administrators can replicate scenarios to test new applications or update the ones that are in use.

During these tests, diverse simulated users were created using different accounts on the GIT repository, working on different branches and collaborating in the creation and update of playbooks and roles. Both the repository of the playbooks and the collaboration is ensured by the way GIT works, providing areas (branches) and versions that can be individual or shared.

Creating more roles and playbooks, as they become needed, will increase the capability of maintaining the applications respecting the norms, standards and processes.

Chapter 5

Conclusions and Future Work

The IST department of large organizations is constantly under a huge amount of work and requests to support the digital work of the users. The department is structured in domains, each with a specific function and in close articulation and collaboration.

The production of applications is fundamental for maintaining the systems working and for dealing with increasing load, derived from the increasing importance of information technologies in the modern society.

On the other hand, although the work, requirements are increasing, there is usually no possibility to also increase the personal, mainly because of budget restrictions. This makes essential the optimization of the activities performed by the IST department and, consequently, the activities performed by the domains that composed it.

One critical domain of the IST department is the Production domain. It is responsible for installing, updating and maintaining the applications in full operation through all the phases and all the necessary environments.

Methodologies typical in industrialization provide a base of standardization and normalization of documentation and processes that lead to increasing levels of productivity. The work presented in this document describes an approach to use the same normalization and standardization processes in the production of applications in large organizations.

The process of developing, purchasing and installing a new application or to update an existing application involves the contribution of many people and teams. It is essential

to facilitate the collaboration through appropriate tools and techniques.

In this work, both domains come together around a CMFs, in this case Ansible, supported by the GIT version control system. This assembly provides a robust and well structured workflow that allows developing, testing and sharing playbooks that can be used by different teams in different environments, through all the installation phases.

All actors from different domains can see the configurations in a form that reflects the infrastructure, that is retracted in the code and configurations on Ansible. The use of GIT workflows enable a controlled collaboration between actors, with traceability and enforces the compliance with standards, norms, and processes. The execution of tasks follows the standards and norms since most of them are automated in playbooks, which follows a specific process.

The use of the template mechanism is also important to maintain consistency between all configuration files in the diverse environments.

In addition, this approach helps to create an updated documentation of the configurations state. It minimizes the gap between the actors that work in different domains, including the project managers, and enables them to view the big picture and the dependencies of an application with their components and underline services and systems.

Considering the tests performed, it is expected that this framework can evolve continually to aggregate new workflows. A subset of the proposed framework is already in use in a large enterprise that deals with a big IT department divided in domains with specialization groups.

As enterprises start to use this approach, some of the routine activities can be executed by the execution of playbooks. As future work, a solution to enable users managing playbooks, request or schedule their execution on priviledged systems may be a possibility.

Bibliography

[1] Ansible. *Overview How Ansible Workks*. https://www.ansible.com/how-ansible-works.

[2] John Arundel. *Puppet 3 Beginner's Guide*. Packt Publishing Ltd, 2013.

[3] Vivre aujourd'hui. *La gestion de projet informatique*. http://www.gestion-projet-informatique.vivre-aujourdhui.fr.

[4] James O Benson, John J Prevost, and Paul Rad. "Survey of automated software deployment for computational and engineering research". In: *10th Annual International Systems Conference, SysCon 2016 - Proceedings*. University of Texas at San Antonio, San Antonio, United States. IEEE, June 2016, pp. 1–6. ISBN: 9781467395182. DOI: 10.1109/SYSCON.2016.7490666. URL: http://ieeexplore.ieee.org/document/7490666/.

[5] Chef. *Chef Overview*. https://docs.chef.io/chef_overview.html.

[6] Chef. *Welcome to Learn Chef*. https://learn.chef.io.

[7] Information Technology Department. *Modèle Dossier d'architecture Technique*. Tech. rep. Available from: https://vdd.coe.int/2.8.2/2_MODELES_DES_LIVRABLES/MOD_ARCHITECTURE_TECH/DAT_ModeleDossierArchitectureTechnique.doc: Council of Europe, 2012.

[8] Kirill Kolyshkin. "Virtualization in linux". In: *White paper, OpenVZ* 3 (2006), p. 39.

[9] Harirajan Padmanabhan and Manjunath Kamath. "Applicability of Industrial Manufacturing Innovations and Concepts for the Industrialization of IT." In: *SRII Global Conference* (2012), pp. 793–802. DOI: 10.1109/SRII.2012.90. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6311067.

[10] Puppet. *How it works*. https://puppet.com/product/how-puppet-works.

[11] Armin Ronacher. *Welcome to Jinja2*. http://jinja.pocoo.org/docs/2.9/.

[12] Saltstack. *Salt in 10 Minutes*. https://docs.saltstack.com/en/latest/topics/tutorials/walkthrough.html.

[13] Daniel Simon and Frank Simon. "IT industrialisation as enabler of global delivery". In: *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*. SQS Software Quality Systems AG, Koln, Germany. IEEE, May 2012, pp. 477–482. ISBN: 9780769546667. DOI: 10.1109/CSMR.2012.62. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6178925.

[14] Mark Stillwell and Jose G F Coutinho. "A DevOps approach to integration of software components in an EU research project". In: *1st International Workshop on Quality-Aware DevOps, QUDOS 2015 - Proceedings*. Imperial College London, London, United Kingdom. New York, New York, USA: ACM Press, Sept. 2015, pp. 1–6. ISBN: 9781450338172. DOI: 10.1145/2804371.2804372. URL: http://dl.acm.org/citation.cfm?doid=2804371.2804372.

[15] Sanjeev Thakur et al. "Mitigating and patching system vulnerabilities using ansible: A comparative study of various configuration management tools for iaas cloud". English. In: *Advances in Intelligent Systems and Computing*. Amity University, Uttar Pradesh, Noida, India. New Delhi: Springer India, Jan. 2016, pp. 21–29. ISBN: 9788132227533. DOI: 10.1007/978-81-322-2755-7_3. URL: http://link.springer.com/10.1007/978-81-322-2755-7_3.